Authoring Intelligent Tutoring Systems for 3D Game Environments

Shrenik Devasani¹, Stephen Gilbert¹, Suhas Shetty¹, Nandhini Ramaswamy¹, Stephen Blessing²

¹ VRAC, Iowa State University, 1620 Howe Hall, Ames, IA 50011, USA
² University of Tampa, 401 W. Kennedy Blvd., Tampa, FL 33606, USA
shrenik@iastate.edu, gilbert@iastate.edu, suhas@iastate.edu, nandhini@iastate.edu, sblessing@ut.edu

Abstract. We describe the design of an authoring tool that is intended to help non-programmers develop game-based intelligent tutoring systems. The tutor-building approach involves the creation of states, both atomic and complex, that help model physical and cognitive states respectively. The tutor-author specifies the parameters associated with each entity in the scenario for each state. The resulting tutor dynamically updates the learner model and automatically assesses the performance of the learner and provides customized feedback.

Keywords: Intelligent tutoring system, game engine, authoring tool, situated tutor

1 Introduction

Game-based intelligent tutoring uses the synergy that exists between digital games and intelligent tutoring systems (ITSs) by combining the high degree of exploration and autonomy that characterizes digital games and the dynamic adaptive instruction supported by ITSs [1]. Recent efforts that use such an approach include an environment used by middle schoolers to learn science concepts [2] and another used by soldiers to learn language and culture issues [3].

The last few decades have seen a change in the nature of teaching in modern universities [4]. Research has shown that active engagement in the learning process by students leads to better learning, higher retention of information and development of skills such as logical thinking and independent decision making. Tutoring using games might provide these benefits as they manage to maintain the user's attention with a feeling of immersion within a simulated environment. Games are a form of reward-based learning and encourage active learning. There has been an increased interest by learning scientists in incorporating games and gaming principles into teaching and learning [5].

ITS researchers have begun exploring how games can be used in intelligent tutors. In some domains, games may be the only possible means of simulating and practicing real world problems. Simulation games are being used extensively in the military for
teaching pilots to fly as well as for training on combat scenarios that would otherwise be extremely dangerous and expensive to train in the field [6]. With the use of simulated environments, aggressive game play can help players relax and balance their aggression [7].

Several game-based ITSs have been developed and customized for military training [8, 9]. Yet, authoring tools that allow non-programmers to develop them are uncommon. Authoring of game-based tutors is challenging due to the inherent domain complexity, the dynamic nature of the environment, the different kinds of feedback required, and the interactions between various non-player entities in the game. In this current work we explore an addition to a tutoring architecture that we have created to enable that architecture to not only tutor in such 3D environments but also to allow for the easy authoring of scenarios and instruction within those environments by non-programmers.

2 Previous Work

The Extensible Problem Specific Tutor (xPST) is an open source ITS authoring tool that supports tutoring within game-engine based synthetic environments [10]. It uses a simple modeling language that promotes authoring by non-programmers. The tutor-author must list the sequence of steps to be performed by the trainee and then describe the feedback associated with each step. We conducted a study that demonstrated that users with minimal programming experience can use xPST to create basic tutors for 3D game environments.

The drawback of this prior approach is that the model created by the tutor-author consisted of goalnodes, or steps to be performed by the learner, rather than states in the game. Though the use of goalnodes makes the tutor-building process very simple and usable by non-programmers, we found that goalnodes by themselves do not have sufficient power to encapsulate all information required to model a dynamic environment. Due to the existence of non-player entities and events that can happen without the trainee’s knowledge, the game’s state can change even without any action being performed by the trainee. The goal of the present work is to eliminate this weakness with the prior approach and to design a method for creating a tutor within 3D environments that allows for the full range of possibilities that exist within such environments, but is still not overly burdensome on the tutor-author.

3 Design

In the present work, we are designing a system that will support soldiers practicing realistic squad-level scenarios. One such scenario that we will use as an example here, is approaching and entering a building that may contain a hostage, an insurgent, and a bomb that needs to be defused.

We have conceptualized the tutoring model associated with such a scenario as composed of a collection of author-defined states. A state can be atomic or complex. An atomic state represents the physical state of the real world (game scenario with all
its entities) at a given instance of time. It is described by the values of the properties of the entities in the scenario (Figure 1). Every property can take a special value called “don’t care” (DC). When a property is assigned this value, it does not come into play when two states are being compared. A complex state is described as a pattern of one or more atomic states, defined using regular expressions. It can be used to represent a cognitive state and also helps model events that happen over time. For example, a trainee enters the complex state “S1 (S2|S3)* S4” when he enters S1, moves to S2 or S3, zero or more times and then finally enters S4. In the example scenario, this might correspond to starting at a location outside the building, approaching the building along a low wall, and then finally entering the building. Although the requirement of regular expressions in the modeling of a complex state makes the system less usable by non-programmers, the power that they offer justifies the trade-off.

![Fig. 1. An atomic state consisting of a learner, insurgent, civilian and a bomb.](image)

Every state (both atomic and complex) is classified as one of the five types: “Start State” (represents the state when the scenario is loaded), “Goal State” (represents a state where the objective of the scenario is achieved), “Failed State” (represents a state where the game is lost, and there is no way back for the learner), “IntermediateCorrect” (represents a correct intermediate state) and “IntermediateIncorrect” (represents an incorrect intermediate state).

In addition to the five states described above, a tutor-author can define a “ResponseState” that can be tied to any of the five kinds of states. When the learner enters a particular state, the ResponseState that is tied to it describes the activities that will be performed automatically by non-player entities in the scenario. For example, when the learner approaches an enemy building, a ResponseState can send reinforcements by forcing existing insurgents to move to the entrance of the building, or spawn new insurgents.

There are three kinds of feedback associated with each state – hints, just-in-time messages and prompts. Hints are displayed when help is requested by the learner (Figure 2). Just-in-time messages are displayed when the learner makes a common mistake that the tutor recognizes. Prompts represent feedback that is neither requested
nor based on incorrect events. They are displayed when the learner enters a particular state.

![Image](image1.png)

**Fig. 2.** An example of a hint presented to the learner: Diffuse the bomb.

When the learner is in a complex state, he would also be in an atomic state. Also, it is possible for the current state of the game to match more than one complex state, when a larger complex state encapsulates smaller ones. For example, the complex state “(S1 S2)* S3” encapsulates the complex state “S2 S3”. Hence, the tutor-author is allowed to assign priorities to each state. Feedback associated with the state with a higher priority is presented first.

Time and speed are critical aspects in military training. The authoring tool allows the dynamic creation and updating of multiple counter variables, in order to track the time spent by the learner in performing specific activities. Counters can be created, started, paused or reset, as required, when the learner enters or leaves a particular state. The values of these counters can be used while checking for specific conditions before giving just-in-time feedback. The values of these counters can also be used as pre-conditions for entering a state and post-conditions for leaving a state.

The tutor maintains a learner model by continuously tracking the skills of the learner. Based on the tasks involved in the scenario, the tutor is associated with a skill set, mapped to skill variables. Every state can be tied numerically to one or more skill variables. When the learner enters a particular state or performs an action, the values of the corresponding skill variables are updated accordingly. For example, a skill variable called “Accuracy” can be defined that keeps track of the percentage of accurate shots fired by the learner. Also, the number of hints or just-in-time error
messages the learner receives in a particular state can affect the values of the skill variables.

3.1 Simulation Engine: Virtual Battlespace 2

We are using Virtual Battlespace 2 (VBS2), a commercial-off-the-shelf, three-dimensional military simulator, based on the game engine Real Virtuality, to develop simulations. VBS2 offers realistic battlefield simulations and delivers a synthetic environment for training teams in arms operations and emergency response procedures. It provides user-defined mission scenarios and real-time scenario management facilities. A learner views the virtual environment from the first-person perspective and can move, interact, and operate just as he or she would, in real life.

Similar to the architecture described in our previous work at creating games-based ITSs [10], we have a Tutor Engine, a Listener module and a Presenter module. The Listener module keeps track of the current state of the game by querying the parameters of all the entities in the scenario. This information is passed on to the Tutor Engine over the network, which matches the current state with author-defined states. The Tutor Engine then sends the appropriate tutoring feedback to the Listener module, which is then presented to the learner through the Presenter module.

Using the simulation engine, scenarios can be pre-defined and the tutor-author can concentrate solely on the task of tutor development and tutoring strategies. The biggest advantage of this architecture is that it can be extended to other game engines such as Unity and BigWorld, by modifying just the Listener and Presenter modules, and retaining the core Tutor Engine.

4 Tutor Authoring Process

The first step is to create the scenario using the VBS2 Mission Editor. This involves simple drag-drop of entities and giving them unique IDs. The names of the properties of each entity in the scenario are defined, along with the possible values that they can take. Next, locations in the scenario that would be of interest from a tutoring point of view, such as buildings, are defined.

Once the scenario is ready, the tutor-author can start defining states and design the tutoring strategy. Consider the simple example scenario, “ClearBuilding”, consisting of the learner located outside Building 5 (B5) and an insurgent, a civilian and a bomb, all located in B5. In order to successfully complete the mission, the learner must do the following:

1. Run towards the nearest wall
2. Approach B5, staying close to the wall at all times
   a. Crouch, if near a window
   b. Enter the door
3. Kill the insurgent
4. Evacuate the building
5. Defuse the bomb

Table 1 describes some of the states that could be defined while designing the tutoring model for a tutor for the above scenario.

Table 1. Possible atomic states for the scenario “ClearBuilding”

<table>
<thead>
<tr>
<th>Entity</th>
<th>Property</th>
<th>Start</th>
<th>NearWindow</th>
<th>InsurgentDead</th>
<th>Diffused</th>
<th>PlayerDead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learner</td>
<td>StateType</td>
<td>Start</td>
<td>Intermediate</td>
<td>Intermediate</td>
<td>Goal</td>
<td>Failed</td>
</tr>
<tr>
<td></td>
<td>IsAlive</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td>Location</td>
<td>‘B5’</td>
<td>window</td>
<td>DC</td>
<td>DC</td>
<td>DC</td>
</tr>
<tr>
<td></td>
<td>Action</td>
<td>DC</td>
<td>DC</td>
<td>DC</td>
<td>DC</td>
<td>DC</td>
</tr>
<tr>
<td>Insurgent</td>
<td>IsAlive</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>DC</td>
</tr>
<tr>
<td></td>
<td>Location</td>
<td>B5</td>
<td>DC</td>
<td>DC</td>
<td>DC</td>
<td>DC</td>
</tr>
<tr>
<td></td>
<td>Action</td>
<td>DC</td>
<td>DC</td>
<td>DC</td>
<td>DC</td>
<td>DC</td>
</tr>
<tr>
<td>Civilian</td>
<td>IsAlive</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>DC</td>
</tr>
<tr>
<td></td>
<td>Location</td>
<td>B5</td>
<td>B5</td>
<td>DC</td>
<td>!B5</td>
<td>DC</td>
</tr>
<tr>
<td></td>
<td>Action</td>
<td>DC</td>
<td>DC</td>
<td>DC</td>
<td>DC</td>
<td>DC</td>
</tr>
<tr>
<td>Bomb</td>
<td>Diffused</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>DC</td>
</tr>
<tr>
<td></td>
<td>Exploded</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>DC</td>
</tr>
<tr>
<td></td>
<td>Location</td>
<td>B5</td>
<td>DC</td>
<td>DC</td>
<td>DC</td>
<td>DC</td>
</tr>
</tbody>
</table>

For the state “NearWindow”, a just-in-time feedback message can defined in the following manner (Figure 3):

JIT {Action != “crouch”}: “You must crouch down when near a window.”
Fig. 3. Just-in-time feedback in the state “NearWindow”, when the learner fails to crouch down: You must crouch down when near a window.

Apart from the atomic states defined in Table 1, we can define a complex state “LearnerConfused”, represented by the regular expression “(Start NearWindow){3,}”. This state represents the learner being in a confused state of mind in which he continuously switches between the states Start and NearWindow, three or more times. Here, the complex state “LearnerConfused” can be assigned a higher priority than the atomic state “NearWindow”.

When the tutor is deployed, the tutor engine continuously queries the parameters of all the entities in the scenario using the unique IDs given to them during the scenario building process. The engine tries to match the current state in the game with an author-defined atomic state. The frequency at which the current state is updated depends on the number of entities in the scenario and the number of states defined by the tutor-author. The tutor engine maintains a stack that stores all the atomic states visited by the learner in last in, first out order, called “History”. When it observes a state transition, it adds the current state to the History. Though the engine continuously updates the current state of the game, it is added to the top of the stack only if it notices a state transition. The History helps recognize complex states that the learner might be in, by finding matches between the author-defined complex states (which are regular expressions) and the components of the History that end with the stack top. The time spent by the learner in a state is also recorded.

5 Conclusions and Future Work

We have described the design of an authoring tool that helps a tutor-author think intuitively while creating a tutor for a synthetic environment. The approach helps model complex tutoring strategies, especially when several non-player entities and objects exist in the scenario. It is interesting to note that the tutor-author need not define state transitions (unlike finite-state automata), since the current state of the game is continuously updated and matched with author-defined states. The ability of a tutor-author to define a property value of an entity as a “don’t care” helps alleviate a possible explosion in the number of states, many of which might be unobservable to both the learner and the tutor-author. Also, though the number of complex states in a realistic environment might reach infinite proportions, the tutor-author needs to define only the subset of the possible states for which he or she wishes to provide appropriate feedback.

Once the authoring tool is completely developed, we would like to conduct an empirical evaluation study, similar to the one described in [10] that demonstrates that the tool is actually feasible for non-programmers to build tutors for game scenarios.

In [10], as a proof of concept, we described how xPST has been adapted to provide support for tutoring on web interfaces, based in part on real-time physiological data. This can be extended to game-based tutors as well, where stress is a major factor that affects a learner’s performance. Variables associated physiological signals such as the
electrocardiogram signal, the heart rate signal and the heart rate variability signal can be used as parameters while defining states in the cognitive model of a tutor.

We would like to further simplify the task of building tutors by employing a technique known as “programming by demonstration” [11]. The tutor-author can build states by demonstrating the task. At any given instance of time during the demonstration, the tutor-author can record the atomic state at that instance, rather than explicitly listing out the values of the parameters, and then specify the feedback associated with that state. The set of complex states can be defined explicitly at a later point in time. This approach will help save a considerable amount of time by giving the tutor-author a head start, compared to having to start from scratch.

When multiple learners are involved in a scenario performing collaborative tasks, a Trainer Observation System can prove to be very handy. It can provide the trainer with useful insight through auto-generated statistics and data visualization features that reflect the performance of the learners, both as a team, and as individuals.

We also plan to design a Learning Management System (LMS) that stores a collection of scenarios and numerical values of the skill variables required for those tasks. As a learner completes tasks in a scenario, the LMS can offer further scenarios according to the learner’s strengths and weaknesses, as reflected in the learner model.

References